Introducción a la programación funcional

Taller de álgebra I

Segundo cuatrimestre de 2015

Resolviendo problemas con una computadora

- La resolución de un problema usando una computadora requiere de al menos los siguientes pasos:
  1. Identificar el problema a resolver, idealmente dando una especificación formal del mismo.
  2. Pensar un algoritmo para resolver el problema.
  3. Implementar el algoritmo en un lenguaje de programación y en una plataforma determinada, obteniendo así un programa ejecutable.

Ejemplo

- **Problema:** Dados dos números naturales representados en notación decimal, encontrar la suma de los números.
- Podemos pensar varios algoritmos para resolver este problema.

1. **Algoritmo escolar**: Sumo las unidades del primero a las del segundo, después las decenas, etc ("llevándome uno de acarreo" cuando hace falta)
2. **Algoritmo sucesor**: Voy sumando 1 al primero y restando uno al segundo, hasta que el segundo llegue a 0.
3. **Algoritmo "rincón del vago"**: Entro a google.com, escribo el primer número, luego el signo "+", luego escribo el segundo y luego aprieto enter.

Ejemplo

- Los tres algoritmos difieren en los pasos primitivos usados para expresarlos:
  1. sumar números de un dígito cada uno, contemplar el acarreo, concatenar resultados, etc.
  2. sumar y restar 1, y comparar contra el 0.
  3. Usar teclado y mouse de una computadora conectada a Internet.
- ¿Cuál algoritmo tiene más sentido/conviene usar?

Programas

- Un programa es la descripción de un algoritmo en un lenguaje de programación.
  1. Es una descripción precisa, de modo tal que pueda ser ejecutada por una computadora.
  2. Un lenguaje de programación tiene una sintaxis y una semántica bien definidas.
- Cuando se implementa un programa, es importante preguntarse
  1. si el programa es correcto,
  2. si implementa adecuadamente el algoritmo propuesto,
  3. si puede pasar que el programa no termine,
  4. qué datos son válidos para ejecutar el programa,
  5. cuánto va a tardar la ejecución,
  6. si está bien resuelto el problema original.

Etapas en el desarrollo de un programa

1. **Especificación**: Es una descripción clara y precisa, idealmente en un lenguaje formal.
2. **Diseño**: Pensar la estructura del programa, dividir la solución en módulos, analizar la interacción entre estos módulos, etc.
3. **Programación**: Escribir el código en un lenguaje de programación.
4. **Validación**: Determinar si el programa cumple con lo especificado.
5. **Mantenimiento**: Corregir errores y adaptar el programa a nuevos requerimientos.

Programación funcional

- Un programa en un lenguage funcional es un conjunto de ecuaciones orientadas que definen una o más funciones.
  En programa.hs:

```
doble x = 2 * x
triple x = 3 * x
```

- La ejecución de un programa en este caso corresponde a la evaluación de una expresión, habitualmente solicitada desde la consola del entorno de programación.

```
Prelude> doble 10
20
```

- La expresión se evalúa usando las ecuaciones definidas en el programa, hasta llegar a un resultado.
- Las ecuaciones orientadas junto con el mecanismo de reducción describen algoritmos (definición de los pasos para resolver un problema).

Programación funcional

Primer Ejercicio: Programar las siguientes funciones

$doble(x) = 2x$

$suma(x, y) = x + y$

$||(v_1, v_2)|| = \sqrt{v_1^2 + v_2^2}$

$f(x) = 8$

$respuesta = 42$

- `doble x = ??`
- `suma x y = ??`
- `normaVectorial v1 v2 = ??`
- `funcionConstante8 x = ??`
- `respuestaATodo = ??`

Ejecutar las siguientes expresiones en el intérprete

```
Prelude> doble 10
Prelude> doble -1
Prelude> suma (-1) 4
Prelude> normaVectorial 3 5
Prelude> funcionConstante8 0
Prelude> respuestaATodo
Prelude> doble 10 20
```

Definiciones de funciones por casos

Podemos usar guardas para definir funciones por casos:

$$f(n) = \begin{cases} 1 & \text{si } n = 0 \\ 0 & \text{si no} \end{cases}$$

```
unoSiCero n  | n == 0 = 1
             | n /= 0 = 0
```

O también...

```
unoSiCero 0 = 1
unoSiCero n | n /= 0 = 0
```

Definiciones de funciones por casos

Ejercicio: Signo

$$signo(n) = \begin{cases} 1 & \text{si } n > 0 \\ 0 & \text{si } n = 0 \\ -1 & \text{si } n < 0 \end{cases}$$

Ejercicios

- Implementar la función `signo`
- Implementar la función `abs` que calcula el valor absoluto de un número ¿Está bueno repetir? ¿Conviene reutilizar?
- Implementar la función `maximo` que devuelve el máximo entre 2 números.
- Implementar la función `maximo3` que devuelve el máximo entre 3 números.

No todo son números

```
esPar x = (div x 2) * 2 == x
esPar 4 ⇝ ? True
esPar 5 ⇝ ? False
```

Alternativa:

```
esPar' x = (mod x 2) == 0
```

Funciones con valores de verdad. True/False

- Los valores `True`/`False` representan valores de verdad, y se suelen usar como valor de retorno de funciones que toman una decisión binaria.
- Ejemplos:

  1. Decidir si un número es positivo.
  2. Decidir si un número es par.
  3. Decidir si un número es primo (aja).
  4. Decidir si un número es perfecto (epa!)[1].
  5. Decidir si dos números son amigos (wtf?)[2].

Ejercicio

Escribir una función `esPositivo` que dice si el parámetro es positivo.

[1] Los primeros cuatro son: 6, 28, 496 y 8128

[2] Por ejemplo 220 y 284

Funciones con valores de verdad. True/False

Llamemos *yLogico*(X, Y) a la función que dados 2 booleanos, nos devuelve su conjunción. Por ejemplo: *yLogico*(Verdadero, Falso) $= V \wedge F = F$.

Ejercicio: Y-lógico

- Implementar la función *yLogico* en Haskell por casos (tabla de verdad)
- Implementar una versión más compacta de la función anterior
- Buscar (en la red de redes) cómo se llama esa función en Haskell.

Notación Prefija vs. Notación Infija

- A veces, las funciones se definen de manera Infija (como `&&` por ejemplo) y a veces las funciones se utilizan de manera Prefija (como `mod` por ejemplo)
- Pero!... podemos utilizarlas como más cómodo nos quede:
  - A infijo: `2 `mod` 3` (utilizando las comillas invertidas)
  - A prefijo: `(&&) True False` (agregando paréntesis)

Ejercicios

1. Programar la siguiente función:

$$f(n_1, n_2, n_3) = \begin{cases} n_1 & \text{si } n_2 < 10 \\ n_1 + n_3 & \text{si } n_2 \geq 10 \end{cases}$$

2. Programar la función $\text{nand}(x, y) = \neg(x \wedge y)$ también por medio de su tabla de verdad.
3. Repetir el ejercicio anterior para la función $\text{nor}(x, y) = \neg(x \vee y)$.
4. Programar una función que tome tres parámetros $a, b, c \in \mathbb{R}$ y que calcule alguna de las raíces de la función cuadrática $f(x) = ax^2 + bx + c$.
5. Programar la función `esPitagorica` que tome tres parámetros `a b c` y diga si existe un triángulo rectángulo donde `a` y `b` sean las medidas de los catetos y `c` la de la hipotenusa.